Correct Reuse of Transformations is Hard to Guarantee

Rick Salay[1], Steffen Zschaler[2], and Marsha Chechik[1]

[1] Department of Computer Science, University of Toronto,
{rsalay|chechik}@cs.toronto.edu
[2] Department of Informatics, King's College London, szschaler@acm.org

Abstract. As model transformations become more complex and more central to software development, reuse mechanisms become more important to enable effective and efficient development of high-quality transformations. A number of transformation-reuse mechanisms have been proposed, but so far there have been no effective attempts at evaluating the quality of reuse that can be achieved by these approaches. In this paper, we build on our earlier work on transformation intents and propose a systematic approach for analyzing the soundness and completeness of a given transformation reuse mechanism with respect to the preservation of transformation intent. We apply this approach to analyze transformation-reuse mechanisms currently proposed in the literature and show that these mechanisms are not sound or complete. We show why providing sound transformation reuse mechanisms is a hard problem, but provide some evidence that by limiting ourselves to specific families of transformations and modeling languages the problem can be simplified. As a result of our exploration, we propose a new research agenda into the development of sound (and possibly complete) transformation reuse mechanisms.

1 Introduction

As model transformations become more complex and more central to software development, automated reuse mechanisms become more important to enable effective and efficient development of high-quality transformations. However, while automating the reuse mechanism is a good first step, it is only useful if it ensures transformations are reused *correctly;* that is, for their intended purpose.

Currently proposed mechanisms for transformation reuse mainly fall into the following two categories: (1) Model typing/sub-typing techniques establish rules for sub-typing relationships between meta-models that can be applied automatically to judge whether a transformation expressed over meta-model MM_A can be executed over meta-model MM_B. Examples of approaches in this area are given in [1,2]. (2) Model concepts and related techniques require that developers wanting to reuse a model transformation defined over meta-model MM_A for meta-model MM_B provide an explicit binding or morphism between the two meta-models. Examples of this approach can be seen in [3,4]. The work of

Pham [5,6] bridges both worlds in that it is based on model typing, but allows the explicit provision of bindings through a dedicated mapping DSL.

In previous work, we have critiqued these approaches and attempted to address some of their shortcomings with respect to correctness. In [7], Zschaler proposes an interface specification approach for correct reuse through modular composition of transformation components. In [8], we argued that correctly reusing a transformation must take into account the *intent* of the transformation. Specifically, a proposed reuse of a transformation can only be considered to be correct if it has the same intent, i.e., it serves the same purpose as the original transformation, albeit in a new context with different kinds of models. We know of no other work attempting to address the question of correct reuse, although the work of Kuehne [9] moves in this direction by giving a theoretical discussion of the varieties of sub-type-like relationships that could support transformation reuse.

In this paper, we explore the theme of transformation intent for correct reuse further. Specifically, we make the following contributions:

- We define the conditions of soundness and completeness of a reuse mechanism with respect to intent and show that existing reuse mechanisms fail to satisfy these conditions.
- We identify reasons why these conditions are difficult to satisfy in general.
- We propose some strategies for mitigating this difficulty by restricting attention to specific families of transformation or modeling languages.

Paper Organization. In Sec. 2, after reviewing the notion of transformation intent, we establish some formal notation and define the key conditions of soundness and completeness of a reuse mechanism. Sec. 3 contains an analysis of existing reuse mechanisms relative to these conditions. In Sec. 4, we discuss the difficulty with satisfying these conditions and then, in Sec. 5, we suggest two strategies for mitigating these difficulties. We conclude in Sec. 6.

2 What is Correct Transformation Reuse?

In this section, we introduce the notion of transformation intent, establish some formal notation for it and then define the key correctness properties of a reuse mechanism with respect to intent.

2.1 Transformation Intent

Informally, we take the *intent* of a transformation to be properties that characterize its general, reusable purpose. For example, consider the transformation $\texttt{minimize} : \texttt{SM} \rightarrow \texttt{SM}$ that minimizes a state machine. 'Minimization' is meaningful also for other model types, so clearly is part of the intent of this transformation. Here we assume that the purpose of a transformation comes from the creator of the transformation and not the user. Thus, the notion of "correct use"

of a transformation is that it is used "as intended". This justifies the notion of "correct reuse" as that of preserving the intent.

In previous work [10], we have cataloged some abstract transformation intents such as *Refactoring*, *Translation*, *Analysis*, etc. that seemed to recur in MDE practice and characterized each intent using a set of properties. That is, every transformation with a given intent must satisfy the corresponding properties. Although our objective in the current paper is not to define abstract intents but instead to capture the intent of particular transformations we wish to reuse, the same approach applies – i.e., we must characterize intent using properties.

Although intent can be characterized by properties, not all properties of transformation are part of its intent. Consider the following properties, listed in order of increasing specialization ($\mathtt{P}_i(f) \Rightarrow \mathtt{P}_{i-1}(f)$), where f is a transformation:

- $\mathtt{P}_1(f) :=$ f preserves well-formedness
- $\mathtt{P}_2(f) :=$ f does model minimization
- $\mathtt{P}_3(f) :=$ f does behavioural model minimization
- $\mathtt{P}_4(f) :=$ f does state machine minimization
- $\mathtt{P}_5(f) :=$ f does minimization using the implication table method [11]

The transformation `minimize` clearly satisfies all five properties. Despite this, only $\mathtt{P}_2$, $\mathtt{P}_3$ and $\mathtt{P}_4$ could be considered to be the intent of `minimize`. Although the general property $\mathtt{P}_1$ holds, it clearly doesn't capture anything significant about the purpose of the transformation. At the other end of the specificity spectrum, $\mathtt{P}_5$ seems to obscure the intent by being too focused on an implementation detail (making it type dependent). These observations suggest the following.

Definition 1 (Transformation intent) *The intent of transformation F is the reusable part of the specification of F that is independent of the type of F.*

The example above also points to the fact that properties that do characterize intent can be ordered in terms of generality. With regard to transformation reuse, we take intent preservation to be defined in terms of the least general common intent between the source and target. For example, if we modify `minimize` to reuse it as $\texttt{minimize}'$ for Labeled Transition Systems (LTS), then we expect $\texttt{minimize}'$ to satisfy $\mathtt{P}_3$ and not the more specific $\mathtt{P}_4$. Similarly, if we are able to find a way to reuse `minimize` for UML Class Diagrams as $\texttt{minimize}''$, it should satisfy $\mathtt{P}_2$ to be a correct reuse.

To our knowledge, no formal criteria have been proposed to distinguish properties that characterize intent from those that do not, and we consider this kind of analysis beyond the scope of this paper. Thus, in this paper, we assume that intent properties have been provided by the transformation developer.

2.2 A Formal Framework

Let Σ be the set of types. For simplicity, we assume that a type $T \in \Sigma$ can either represent a set of models (i.e., the usual idea of a type) or a metamodel

defining such a set of models and what we mean should be clear from the context. Furthermore, Σ includes simple types, product types, function types, etc. as needed. Let Ω be the set of transformations of interest and let the function $type : \Omega \to \Sigma$ assign a type to each transformation. Note that we treat a transformation as a function rather than as a program that implements a function so two transformations that have the same I/O behaviour are the same transformation. Without loss of generality we limit our discussion to transformations with single input and output types. Thus, we assume that for all $F \in \Omega$, $type(F)$ has form $T \to T_1$ where each of T and T_1 are types. For the purposes of this paper, we define a transformation reuse mechanism as follows:

Definition 2 (Transformation reuse mechanism) *A* transformation reuse mechanism *R is a tuple $\langle \mathcal{M}, src, tgt, \rho \rangle$ where $\mathcal{M}$ is a set of specifications called* type mappings*, functions $src, tgt : \mathcal{M} \to \Sigma$ extract the source and target types of a mapping and $\rho : \Omega \times \mathcal{M} \to \Omega$ is a partial function called the* reuse transformation *such that if $\rho(F, M)$ is defined then $src(M) = type(F)$ and $tgt(M) = type(\rho(F, M))$.*

Thus, we reuse a transformation F for a new transformation type $T' \to T'_1$ by supplying type mapping M having the new type as the target and then computing $\rho(F, M)$ to produce the new transformation. Note that since ρ is a partial function, it can limit the possibilities for reuse. For example, if $\rho(F, M)$ is not defined for any $M \in \mathcal{M}$ where $tgt(M) = T' \to T'_1$, then we interpret this as the reuse mechanism asserting that it cannot be used to reuse F for this new type.

Definition 3 (Intent order) *The* intent order *$\langle \Psi, \preceq \rangle$, where Ψ is the set of transformation intents of interest, is a partial order such that for every pair of intents $I_1, I_2 \in \Psi$, if they have a common upper bound then there exists a least upper bound designated $I_1 \vee I_2 \in \Psi$. The function $intent : \Omega \to \Psi$ assigns to each transformation its most specific intent in Ψ. The relation $\sim \subseteq \Omega \times \Omega$ called* transformation similarity *satisfies the condition that for all transformations, $F, F' \in \Omega$, $F \sim F'$ iff $intent(F) \vee intent(F')$ exists.*

The ordering relation $\preceq$ captures the generalization hierarchy of intents where $I_1 \preceq I_2$ means that I_2 is a more general intent than I_1. Thus, in our example above, $\mathtt{P_4} \preceq \mathtt{P_3} \preceq \mathtt{P_2}$. Note that not all pairs of intents typically have a common upper bound and in particular, we assume the intent order has no top element. To compare intents of different transformations we use a transformation similarity relation. Thus, two transformations are considered similar if they share the same intent at some level of generality. We can now define some key properties of a transformation reuse mechanism using transformation similarity.

Definition 4 (Sound and complete reuse) *Let $R = \langle \mathcal{M}, src, tgt, \rho \rangle$ be a transformation reuse mechanism and $\langle \Psi, \preceq \rangle$ be the intent order. We define the following properties of R:*

- *(soundness) R is a* sound *reuse mechanism iff for all $F \in \Omega, M \in \mathcal{M}$, if $\rho(F, M)$ is defined then $F \sim \rho(F, M)$.*

– *(strong completeness) R is a* strongly complete *reuse mechanism iff for all $F, F' \in \Omega$, if $F \sim F'$ then there exists $M \in \mathcal{M}$ such that $\rho(F, M) = F'$.*
– *(weak completeness) R is a* weakly complete *reuse mechanism iff for all $F, F' \in \Omega$, if $F \sim F'$ then there exists $M \in \mathcal{M}$ such that $type(\rho(F, M)) = type(F')$.*

Soundness says that the reuse transformation always preserves intent. The reuse mechanism is strongly complete when every transformation that shares an intent with F can be a reuse target. It is weakly complete when for every transformation type with a transformation sharing an intent with F, there is a reuse target with this type. Note that soundness and completeness is relative to the choice of Ω.

Since we understand the correct reuse of a transformation to mean that it shares an intent with the original transformation, a reuse mechanism must be sound to *guarantee* correct reuse. In addition, it can be complete, under either definition of completeness. Completeness concerns the scope of applicability of a reuse mechanism (and thus the flexibility it offers). For example, a reuse mechanism that can only allow the reuse of the identity transformation (i.e., makes no change to the input) on a model type is trivially sound but its incompleteness makes it useless in practice. At the other end of the spectrum, a reuse mechanism that allows arbitrary Java "adapter code" to be specified to change the behaviour of a given transformation is trivially complete but is clearly not sound since one can write an adapter to use any transformation in ways it was not intended.

3 Analysis of Some Existing Transformation Reuse Mechanisms

In this section, we analyze existing transformation reuse approaches in the literature and assess whether they satisfy the soundness and completeness conditions defined in Sec. 2. We focus on the two main classes of work reviewed in the Sec. 1 and conclude with general comments about other approaches.

3.1 Model Typing

The paper "On Model Typing"[1, later refined in [2]] is one of the earliest proposals for transformation reuse based on *sub-type substitutability* using the following argument: Let $F : T_1 \rightarrow T$ be a transformation and T_1' be some model type. Then we can reuse F without alteration, on inputs of type T_1' iff some type matching condition $match(T_1', T_1)$ holds [1].

Fig. 1 shows five state machine metamodels reproduced from [1]. Metamodel `M0` is the base type and the remaining four are variants of this. According to the type matching condition, all variants except `M1` with multiple start states satisfy the matching condition. The authors argue that if a transformation written for `M0` navigated the `initialState` reference it would expect (at most) one `State`

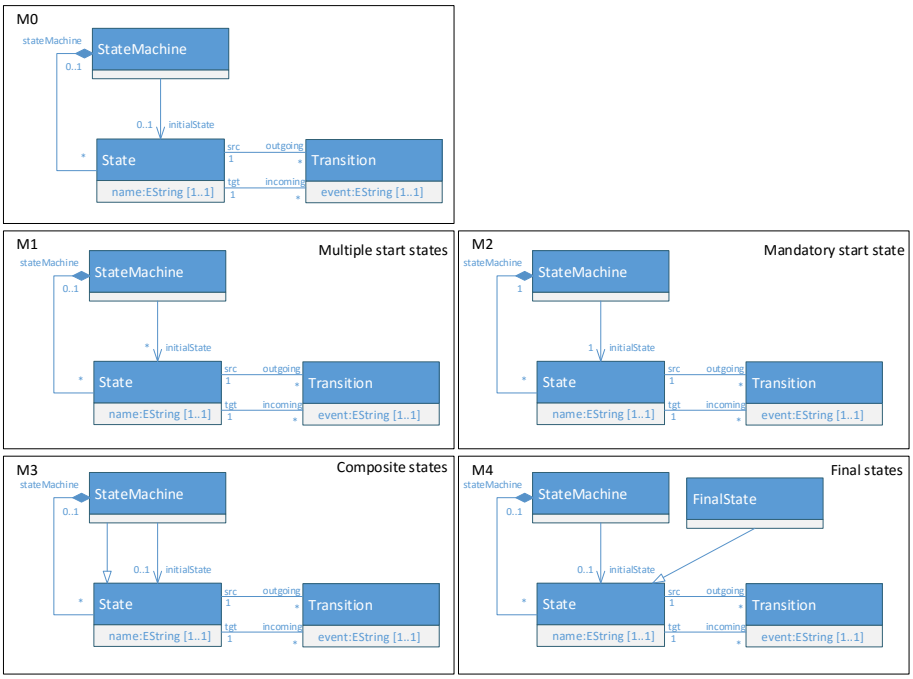

Fig. 1. Five state machine variants: `M0`:Base; `M1`:Multiple start states; `M2`:Mandatory start state; `M3`:Composite states; `M4`:Final states.

object but if it was reused with models of type `M1` then it could find multiple `State` objects and thus "break".

Analysis. Using Defn. 2, let $\mathtt{R_1} = \langle \mathcal{M}_1, src_1, tgt_1, \rho_1 \rangle$ be the reuse mechanism used in this approach and assume we are reusing transformation $F : T_1 \to T$ for new input type T_1'. According to the definition of type matching used, no additional mapping information is required other than the metamodels of the source and target types T_1 and T_1'. Thus, we let $\mathcal{M}_1 = \Sigma \times \Sigma$ store the source and target types and define src_1 and tgt_1 to extract the first and second components of this pair, respectively. Finally, $\rho_1(F, \langle T_1, T_1' \rangle)$ is defined iff $match(T_1', T_1)$ holds and when it is defined, $\rho_1(F, \langle T_1, T_1' \rangle)$ is the same transformation as F but restricted to inputs of type T_1'. Thus, $type(\rho_1(F, \langle T_1, T_1' \rangle)) = T_1' \to T$ as required.

We now show that $\mathtt{R_1}$ is both unsound and incomplete.

While no formal specification is given, the transformation given as an example in the paper is described as follows: "Takes as input a state machine and produces a lookup table showing the correspondence between the current state, an arriving event, and the resultant state." We take this to be our designated transformation F. Since no other statement about the intent of the transformation was given, we considered two possible intents:

(I1) To produce a tabular representation of all the state-state transitions.

(I2) To produce a tabular representation of the state machine (i.e., it encodes the same set of traces but as a table).

Intent I1 is a direct restatement of the definition of the transformation while I2 attempts to abstract from the implementation to capture the underlying purpose of the transformation.

The intent is preserved for all instances except variant `M3` with intent I2. The transformation reuse fails to preserve intent here because variant `M3` has composite states and the simple algorithm given in the transformation definition ignores the containment relation between states and hence cannot fully capture its semantics.

Thus, for intent I2, we have that $\rho_1(F, \langle \texttt{M0}, \texttt{M3} \rangle)$ is defined but $F \not\sim \rho_1(F, \langle \texttt{M0}, \texttt{M3} \rangle)$. Thus, by Defn. 4, $\texttt{R}_1$ is unsound. To show that it is also incomplete, we must find a transformation F' that does preserve intent while not being producible via ρ_1. Recall that variant `M1` is an example of a state machine that *does not* satisfy *match* and so $\rho_1(F, \langle \texttt{M0}, \texttt{M1} \rangle)$ is not defined. However, it is clear that both intents I1 and I2 *would be preserved* if a transformation F' defined as given for F was applied to variant `M1`. That is, a plausible algorithm for producing the table could be created by enumerating through the transitions and never navigating the `InitialState` reference. Thus, the issue of multiplicity would not affect this algorithm. This shows that $\texttt{R}_1$ is incomplete and that the definition of *match* is unduly restrictive.

Guy *et al.* [2] have extended the work of Steel and have proposed more sophisticated approaches to sub-typing based reuse; however, all of their proposals can be shown to be unsound with respect to intent following a similar line of reasoning as given above. Only one of their proposals, *non-isomorphic subtyping*, may be complete, because it allows arbitrary adaption transformations to be applied to models of type T_1' before being given as input to F. Thus, in principle, F could be reused for *any* input type this way. The cost of allowing this flexibility is to make soundness even more difficult to guarantee.

3.2 Model concepts

The main alternative approach to transformation reuse has been proposed by Rose, de Lara, *et al.* in [4,12]. Their approach is based on the definition of explicit bindings between elements of a concrete meta-model and a so-called concept (an abstract meta-model that the transformation to be reused is defined over) instead of a generic type mapping.

Analysis. At first, it may appear that this resolves the problem we have identified in Sec. 3.1: after all, to reuse a transformation and its intent, we only have to define a suitable binding that will ensure intent preservation. However, Rose and de Lara's work defines a set of conditions that characterise "valid" bindings. We should then ask whether these conditions are sound or complete wrt intent preservation. We, again, note that in their papers they do not actually say explicitly what intents should be preserved by the transformation reuse.

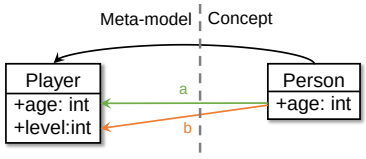

Fig. 2. Example of two different valid bindings of the same meta-model and a model concepts.

Hence, following Defn. 2, let $\mathtt{R}_2 = \langle \mathcal{M}_2, src_2, tgt_2, \rho_2 \rangle$ be the reuse mechanism used in this approach and assume we are reusing transformation $F : T_2 \to T$ for new input type T_2'. A type mapping consists of a source and target meta-model as well as a set of bindings (essentially, a morphism between the two). Thus, we let $\mathcal{M}_2 = \Sigma \times \Sigma \times (\Sigma \to \Sigma)$ to store the source and target types and the binding and define src_2 and tgt_2 to extract the first and second components, respectively. $binding : \mathcal{M}_2 \to (\Sigma \to \Sigma)$ is defined to be a function extracting the last component. Finally, $\rho_2(F, M)$ is defined iff $src_2(M) = T_2$ and $tgt_2(M) = T_2'$ and $binding(M)$ is valid as per the rules for validity defined in [4,12]. When it is defined, $\rho_2(F, M)$ is the same transformation as F but with a coercion from T_2' models to T_2 models injected before the transformation execution. This coercion is generated as described in [12]. Thus, $type(\rho_2(F, M)) = T_2' \to T$ as required.

An important point to note is that the conditions for the validity of bindings are completely syntactic and may in fact allow a range of different concrete bindings between the same concrete meta-model and concept. Fig. 2 shows an example of this. Both the green binding labeled `a` and the orange binding labeled `b` would be valid based on the rules given in [4,3]. They would both lead to syntactically correct transformations preserving syntactic properties such as, for example, the preservation of wellformedness rules. However, they would clearly lead to very different transformation semantics (and intents) of the reused transformations. So, generally, we would expect $\mathtt{R}_2$ to be unsound because it is too flexible.

Rose *et al.* [4] give a different example, which focuses on a single specific binding: they introduce a `TokenHolder` concept to represent abstractly the key concepts in Petri-Net-like modelling languages. They then define a number of transformations (and, in fact, model management operations) over this concept and proceed to show how these can be reused over a proper Petri-Net model as well as models of production-line systems (using a simple type mapping M_C binding parts to tokens, conveyors etc. to holders and machines to processes). An interesting transformation that they discuss is one that refactors `TokenHolder` models by removing any `Process` elements connecting the same set of `Holder` elements. If we analyse this transformation in a similar way to Sec. 3.1, we can identify two possible intents:

(I3) The transformation keeps the syntactic structure of the token holder model, but removes syntactically duplicate elements;
(I4) The transformation maintains the observable semantics of the token holder model.

While the former intent is trivially preserved by the reuse mechanism, the situation is less obvious for the latter intent. For example, for production-line models removing a machine might remove important options for processing parts. Even though machines might take parts from the same trays and place processed parts onto the same conveyors they may actually do very different things with these parts. Additionally, in a production-line design multiple machines feeding from the same trays and sending parts to the same conveyors might be an important performance or reliability optimisation. Thus, for intent I4 we have that $\rho_2(F, M_C)$is defined, but $F \not\sim \rho_2(F, M_C)$. Thus, by Defn. 4, $\mathtt{R}_2$ is unsound.

3.3 Other transformation reuse approaches

So far, we have focused on "black box" reuse mechanisms, which aim to enable reuse of a transformation without deep knowledge of its implementation. Other techniques have been explored for transformation reuse. Kusel *et al.* [13,14] provide a good overview and empirical evaluation of some of these approaches.

"Black box" approaches can be extended to external transformation composition by transformation chaining [15]. Here, the transformation is often executed over models instantiating exactly the same meta-models over which the transformation was defined. As a result, the semantics of the transformation does not change at all, and the original transformation intents are trivially preserved. However, while this makes the reuse mechanism sound, it also makes it quite incomplete as there are potentially a large number of similar meta-models for which the transformation intents could be preserved, but for which the transformation cannot be reused. This insight has been the driver behind the work on model typing and model concepts [1,2,3,4,12].

More invasive, "white box" compositions of transformations (*e.g.,* [16]) are expected to change the intent. However, we would likely want to retain some control over which parts of the intent should be preserved. A preliminary attempt to address this problem through the notion of parameterized transformation semantics has been given in [17], but more work is required.

4 Why Intent Preservation is Hard to Achieve

In the previous section, we showed that soundness (i.e., intent preservation) is a difficult goal to achieve for a reuse mechanism. In this section, we discuss why this is the case.

As we discussed in Sec. 2, a natural way to characterize intent formally is as a property that the transformation must satisfy. In that case, all transformations satisfying this property would also carry the same intent. Then, $\rho(F, M)$ with mapping M is a correct reuse of F iff $P_I(F) \Rightarrow P_I(\rho(F, M))$, where P_I is the property that characterizes the intent of F. In [8], we explored this simple hypothesis and found a flaw – that it is typically not possible to find a single property that is checkable across different transformations.

For example, consider the transformation $\texttt{minimize} : \texttt{SM} \rightarrow \texttt{SM}$ that produces a state machine with the same semantics as the input state machine but with a minimum number of states. We could define this intent by property:

$$\texttt{P}^{\texttt{SM}}_{min}(f) := \forall x : \texttt{SM}\cdot\texttt{Bisim}_{\texttt{SM}}(x, f(x)) \wedge (\forall x' : \texttt{SM}\cdot\texttt{Bisim}_{\texttt{SM}}(x', f(x)) \Rightarrow |x'| \geq |f(x)|)),$$

where f is the transformation, $\texttt{Bisim}_{\texttt{SM}}$ is the predicate that checks bisimilarity, and the size function $|\ |$ returns the number of states.

Let $\texttt{minimize}' = \rho(\texttt{minimize}, \texttt{M})$ represent a reuse of $\texttt{minimize}$ for LTSs. That is, $\mathit{type}(\texttt{minimize}') = \texttt{LTS} \rightarrow \texttt{LTS}$. If ρ is sound, we expect it to preserve intent and thus preserve such characteristic properties. However, this preservation condition cannot be expressed simply as the requirement that $\texttt{P}^{\texttt{SM}}_{min}(\texttt{minimize}) \Rightarrow \texttt{P}^{\texttt{SM}}_{min}(\texttt{minimize}')$. The reason is that $\texttt{P}^{\texttt{SM}}_{\texttt{min}}$ is defined with respect to type $\texttt{SM} \rightarrow \texttt{SM}$ but since the type of $\texttt{minimize}'$ is $\texttt{LTS} \rightarrow \texttt{LTS}$, the expression $\texttt{P}^{\texttt{SM}}_{min}(\texttt{minimize}')$ is not *well-defined*. What we really need is a definition for the property $\texttt{P}^{\texttt{LTS}}_{min}$ that characterizes the intent of minimization for LTSs and then prove that $\texttt{P}^{\texttt{SM}}_{min}(\texttt{minimize}) \Rightarrow \texttt{P}^{\texttt{LTS}}_{min}(\texttt{minimize}')$.

In [8], we proposed an approach for producing $\texttt{P}^{\texttt{LTS}}_{min}$ using *parameterized properties* to characterize intent. In the case of minimization, we define the parameterized property as follows:

$$(1)\quad \texttt{P}^{\langle T\rangle}_{min}(f) := \forall x : T \cdot \texttt{SemEq}_T(x, f(x)) \wedge (\forall x' : T \cdot \texttt{SemEq}_T(x', f(x)) \Rightarrow \texttt{Size}_T(x') \geq \texttt{Size}_T(f(x))),$$

where f is the transformation, $\texttt{SemEq}_T$ is the semantic equivalence relation for models of type T and $\texttt{Size}_T$ is a function that measures the relevant size attribute of models of type T. In our example, we get $\texttt{P}^{\texttt{LTS}}_{min}$ by providing predicates $\texttt{SemEq}_{\texttt{LTS}}$ (i.e., bisimilarity for LTSs) and $\texttt{Size}_{\texttt{LTS}}$ to get:

$$\texttt{P}^{\texttt{LTS}}_{min}(f) := \forall x : \texttt{LTS} \cdot \texttt{SemEq}_{\texttt{LTS}}(x, f(x)) \wedge (\forall x' : \texttt{LTS} \cdot \texttt{SemEq}_{\texttt{LTS}}(x', f(x)) \Rightarrow \texttt{Size}_{\texttt{LTS}}(x') \geq \texttt{Size}_{\texttt{LTS}}(f(x)))$$

We can now state a general procedure for ensuring that $\rho(\texttt{F}, \texttt{M})$ is a correct reuse of transformation $\texttt{F} : \texttt{TT} \rightarrow \texttt{TT}_1$:

1. Provide a parameterized property $P^{\langle T, T_1\rangle}$ that characterizes the intent of F in terms of one or more type-specific predicates $Q_i^{\langle T, T_1\rangle}, i = 1 \ldots n$.
2. For each potential reuse target type $\texttt{TT}' \rightarrow \texttt{TT}'_1$,
   (a) provide the type-specific predicates $Q_i^{\langle \texttt{TT}', \texttt{TT}'_1\rangle}, i = 1 \ldots n$; and
   (b) prove that $\texttt{P}^{\langle \texttt{TT}, \texttt{TT}_1\rangle}(\texttt{F}) \Rightarrow \texttt{P}^{\langle \texttt{TT}', \texttt{TT}'_1\rangle}(\rho(\texttt{F}, \texttt{M}))$.

Now we can see why showing that a reuse mechanism is sound is difficult. To do so, we would have to show that whenever we reuse a transformation using the mechanism, the proof obligation in step (2b) is guaranteed to be satisfied. For automation, this requires theorem proving support and is, in general, undecidable. We observe that techniques similar to those for showing valid refinements may be usable and we discuss this briefly in Sec. 6. However, in addition to

this, steps (1) and (2a) require type-specific information to be provided for each source and target type in a reuse case. For example, this information could be provided as part of the mapping M.

Our conclusion is that while soundness of a reuse mechanism is a desirable goal, achieving it *for the general case* may not be practical. In the next section, we consider special cases where soundness can be guaranteed.

5 Sound Reuse Strategies

In this section, we give preliminary proposals of two strategies that could help achieve sound transformation reuse.

5.1 Reuse across Transformation Families

Our view of intent similarity using $\sim$ suggests that sound reuse occurs with respect to more general intents that are shared by many transformations. For example, the property $\mathtt{P}_{min}^{\langle T \rangle}$ in Eq. 1 is an intent shared by many minimization transformations. This observation points to the following potential strategy for sound reuse: define generic transformations that implement the general intent and instantiate these transformations to produce specific concrete transformations with a more specialized intent. We illustrate this idea using the model minimization intent.

Equation Eq. 1 above gives a parameterized property describing the intent of minimize transformations. This reflects the fact that, in the most abstract case, doing minimization requires a partial order relation (here, $\geq$ over sizes is measured by $\mathtt{Size}_T$) and checking semantic equivalence requires an equivalence relation (implemented by predicate $\mathtt{SemEq}_\mathtt{T}$). We can use these to define an abstract minimization transformation $\mathtt{minimize}^{\langle T \rangle}$ shown in Fig. 3. This simple algorithm enumerates all models of type T with size smaller than the input model X (lines 1-6) until it finds one that is semantically equivalent to X and returns it (line 4). The algorithm is guaranteed to terminate assuming that $\mathtt{Size}_T(X)$ is bounded by the number of elements in X, and S in line 2 only includes non-isomorphic models.

The algorithm is clearly naive; however, we can instantiate it for any model type that can provide the parameters. Furthermore, *each such transformation instance clearly must satisfy the minimization intent property* $\mathtt{P}_{min}^{\langle T \rangle}$. This fact ensures that every reuse of $\mathtt{minimize}^{\langle T \rangle}$ is sound in the sense of Defn. 4.

Generalizing the approach. We can view this transformation family approach to reuse as a reuse mechanism according to Defn. 2. Let $\mathtt{R_{tf}} = \langle \mathcal{M}_\mathtt{tf}, src_\mathtt{tf}, tgt_\mathtt{tf}, \rho_\mathtt{tf} \rangle$ be the *transformation family-based* reuse mechanism. We assume that $\mathtt{R_{tf}}$ is limited to the reuse of parameterized transformations $F^{\langle T \rangle}$ that are, as with the case of $\mathtt{minimize}^{\langle T \rangle}$, implemented by refining the definition of the corresponding parameterized property characterizing the intent of F [3]. To reuse such a parameterized transformation for a specific type $\mathtt{T}'$, a mapping $M \in \mathcal{M}_\mathtt{tf}$ must be

[3] We leave details of this refinement procedure for future work.

Algorithm: Minimize model
Params: $\langle \texttt{Size}_T, \texttt{SemEq}_T \rangle$
Input: Model $X : T$
Output: Minimal model $X' : T$

```
1: for (i = 0 to Size_T(X)) do
2:     Let S = {Y | Size_T(Y) = i}
3:     for (Z ∈ S) do
4:         if SemEq_T(X, Z) then return Z
5:     endfor
6: endfor
7: return X
```

Fig. 3. Algorithm of abstract model minimization transformation $\texttt{minimize}^{\langle T \rangle}$.

supplied consisting of a set of $\texttt{T}'$-specific predicates corresponding to the formal parameters of $F^{\langle T \rangle}$ as well as the metamodel $\texttt{T}'$ itself. The functions $src_{\texttt{tf}}$ and $tgt_{\texttt{tf}}$ extract $T \to T$ and $\texttt{T}' \to \texttt{T}'$, respectively, from the mapping. Finally, $\rho_{\texttt{tf}}$ performs the instantiation operation that generates the specific concrete transformation. Thus, $F^{\texttt{T}'} = \rho_{\texttt{tf}}(F^{\langle T \rangle}, M)$ is obtained by substituting the formal parameters of $F^{\langle T \rangle}$ for their values given in M.

The soundness of $\texttt{R}_{\texttt{tf}}$, according to Defn. 4 follows directly from the fact that the parameterized transformation $F^{\langle T \rangle}$ is designed to be sound for *any* substitution of the type-specific predicates, as long as the given predicates are correct for the type. Thus, $F^{\langle T \rangle} \sim \rho_{\texttt{tf}}(F^{\langle T \rangle}, M)$ as required.

Discussion. The transformation family approach can be seen as a special case of Generic Programming [18] – a technique in which parts of a concrete algorithm are abstracted as parameters to an abstract algorithm. Since the model-concepts approach to reuse discussed in Sec. 3.2 also cites generic programming as an inspiration, it is important to discuss why that approach fails our soundness test whereas the transformation family approach succeeds.

With model concepts, the genericity comes from defining the reusable transformation relative to a generic meta-model. The notion corresponding to instantiation is the binding from the concrete meta-model to the generic one. Since this binding is limited to meta-models, it is purely syntactic (and so are the conditions on 'valid' bindings) and there is no way to access richer type-specific information – such as, for example, conditions for semantic equivalence – as part of the instantiation process. Thus, for a certain class of purely syntactic transformations, the model concepts approach may be sound, but for general transformations, the additional semantic coherence of the transformation family approach is required to ensure soundness.

5.2 Reuse across Model Type Families

In the transformation family approach to reuse, we showed how generic parameterized transformations could be developed by expressing intent using parameterized properties. While this technique provides a sound reuse strategy, much of the reuse complexity may be pushed into the type-specific parameters. For example, for $\texttt{minimize}^{\langle T \rangle}$, the predicate $\texttt{SemEq}_T$ may be non-trivial to define. As

discussed in Sec. 4, this undermines the quality of "effort reduction" implied by a reuse mechanism. A second issue is that while generic parameterized transformations may be broadly applicable, they may not be able to exploit the efficiencies available to specific classes of model types. In the special case of transformation reuse across a family of closely-related model type variants, it may be possible to mitigate these problems.

We illustrate these issues with the example of state machine variants discussed in Sec. 3.1. Assume that the parameters for the base state machine variant `M0` in Fig. 1 are defined as follows:

- $\texttt{SemEq}_{\texttt{M0}}(X, X')$ holds iff state machine X is bisimilar to X'.
- $\texttt{Size}_{\texttt{M0}}(X)$ is defined as the number of states in state machine X.

We can think of the state machine variants as forming a model type family with the base variant `M0` as the core representative. All our variants are state machines and are based on the same semantic interpretations as `M0`. Thus, semantic equivalence for any variant is still defined as bisimilarity. Furthermore, state machine size, for the purposes of minimization, is also defined as the number of states for all variants. Thus, the parameters of the minimization intent property $\texttt{P}_{min}^{\langle T \rangle}$, for any model type in this family, are the same as those for `M0`, allowing these parameters to be defined only once for the entire family. If we use the parameterized transformation $\texttt{minimize}^{\langle T \rangle}$, we can also use the common model type parameter values for any member of the family to instantiate $\texttt{minimize}^{\langle T \rangle}$ – the only parameter that varies in each case is the metamodel of the model type. Thus, using the state machine family reduces reuse effort while retaining soundness.

Now since the only part of the transformation that varies among all the members of the state machine family is the metamodel of the particular state machine type to which it is applied, this raises the question of whether the transformation algorithm itself can be refined for the state machine family. For example, consider the *implication table method* [11] – a minimization method designed specifically for (finite) state machines. A more efficient minimization transformation using this could be developed to implement the intent "state machine minimization" rather than the general intent "model minimization". Furthermore, if we can show that this transformation is semantically equivalent to the instantiation of our naive transformation in Fig. 3, then we have created a more efficient transformation that is soundly reusable within the state machine family.

Generalizing the approach. The model type family strategy is not a reuse mechanism itself but rather a way to mitigate some of the difficulties with the transformation family strategy. As a result, it can be viewed as an extension of the transformation family reuse mechanism.

Assume we wish to reuse the parameterized transformation $F^{\langle T \rangle}$. We summarize the steps of the model type family strategy as follows. First we define a model type family over which all the parameters of $F^{\langle T \rangle}$, except the metamodel of T, have the same value. This makes the instantiation of $F^{\langle T \rangle}$ across the model

type family dependent only on the metamodel, i.e., we have effectively the same transformation that works for any member of the model type family. Second, now that we know that a transformation that can be soundly reused across the family exists, we consider whether we can define a more efficient transformation, that is semantically equivalent to this one.

Discussion. The model type family strategy combined with transformation families provides two sources of benefits. First, effort is reduced because the same parameter values can be used to instantiate the transformation for any member of the family. Second, the commonalities in the family can point to more refined and efficient generic implementations of a transformation. While there exist methods for defining families of models (e.g., product lines), defining one for which all members have the same parameter values clearly may be non-trivial, and we consider methods for this to be future work.

6 Conclusion

In this paper, we investigate the need for model-transformation reuse mechanisms to ensure correctness of transformation reuse. Specifically, we define a formal framework for the analysis of transformation reuse mechanisms with respect to intent preservation. We define reuse mechanisms as consisting of mappings between modeling languages that induce a translation of model transformations. We say that a reuse mechanism is *sound* if it preserves the intent of all transformations it translates and it is *complete* if it can produce all intent-preserving translations of a transformation. We then showed that none of the currently proposed transformation reuse mechanism are sound and that completeness is only currently achieved by completely ignoring model semantics through allowing arbitrary adaptations of transformations.

To address this gap, we began by showing that correct transformation reuse is hard to guarantee because it requires verifying non-trivial semantic properties across modeling languages. We then showed that we could ensure correct reuse of a transformation if it was derived from a formal expression of intent and parameterized with language-specific information. While this provides a sound reuse mechanism, it may not lead to reused transformations that are efficient. In a next step, we showed that limiting transformation reuse to a semantically coherent family of modeling languages can further simplify the problem and allow for efficient transformations.

Our work brings us one step closer to an "algebra of model management" by providing the formal basis for studying transformation reuse. We invite the research community to help us in working on this research agenda and answering the following research questions: 1) How can intents be described effectively? We have explored the use of parameterized formalizations, but have also indicated the need for limiting the genericity of transformation intents. Consequently, there is a need for a formal language for expressing transformation intents in a precise manner. 2) What are the precise sufficient conditions for simple, correct transformation reuse within families of modeling languages? 3) Can we define a *reuse*

calculus for specific classes of intents and transformations that enables coupling the refinement of intents and the refinement of transformation implementations? 4) What are mechanisms and languages for constructing concrete, sound (and possibly complete) reuse mechanisms from descriptions of classes of intents?

References

1. Steel, J., Jézéquel, J.M.: On Model Typing. SoSyM **6**(4) (2007) 401–413
2. Guy, C., Combemale, B., Derrien, S., Steel, J.R., Jézéquel, J.M.: On Model Subtyping. In: Proc. of ECMFA'12. (2012) 400–415
3. de Lara, J., Guerra, E.: From Types to Type Requirements: Genericity for Model-Driven Engineering. SoSyM **12**(3) (2013) 453–474
4. Rose, L., Guerra, E., de Lara, J., Etien, A., Kolovos, D., Paige, R.: Genericity for Model Management Operations. SoSyM (2011)
5. Pham, Q.T., Beugnard, A.: Automatic Adaptation of Transformations Based on Type Graph with Multiplicity. In: Proc. of SEAA'12. (2012) 170–174
6. Pham, Q.T.: Model Transformation Reuse: A Graph-based Model Typing Approach. PhD thesis, Université de Rennes (2012)
7. Zschaler, S.: Towards Constraint-Based Model Types: A Generalised Formal Foundation for Model Genericity. In: Proc. of VAO'14. (2014)
8. Salay, R., Zschaler, S., Chechik, M.: Transformation Reuse: What is the Intent? In: Proc. of AMT@MODELS'15. (2015) 7–15
9. Kühne, T.: On Model Compatibility with Referees and Contexts. Software & Systems Modeling **12**(3) (2013) 475–488
10. Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G.M., Syriani, E., Wimmer, M.: Model Transformation Intents and Their Properties. SoSym (2014) 1–38
11. Paull, M.C., Unger, S.H.: Minimizing the number of states in incompletely specified sequential switching functions. Electronic Computers, IRE Transactions on (3) (1959) 356–367
12. de Lara, J., Guerra, E.: Towards the Flexible Reuse of Model Transformations: A Formal Approach Based on Graph Transformation. J. Logical and Algebraic Methods in Programming **83**(5–6) (2014) 427–458
13. Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W.: Reuse in Model-to-Model Transformation Languages: Are We There Yet? SoSyM **14**(2) (May 2015) 537–572
14. Kusel, A., Schönböck, J., Wimmer, M., Retschitzegger, W., Schwinger, W., Kappel, G.: Reality Check for Model Transformation Reuse: The ATL Transformation Zoo Case Study. In: Proc. AMT@MODELS'13. (2013)
15. Vanhooff, B., Ayed, D., Baelen, S.V., Joosen, W., Berbers, Y.: UniTI: A Unified Transformation Infrastructure. In: Proc. of MODELS'07. Volume 4735 of LNCS. (2007) 31–45
16. Wagelaar, D., van der Straeten, R., Deridder, D.: Module Superimposition: A Composition Technique for Rule-Based Model Transformation Languages. SoSyM **9** (2010) 285–309
17. Zschaler, S., Terrell, J., Poernomo, I.: Towards Modular Reasoning for Model Transformations. In: Workshop on Composition and Evolution of Model Transformations, King's College London, Dept. of Informatics. (2011)
18. Musser, D.R., Stepanov, A.A.: Generic Programming. In: Symbolic and Algebraic Computation. Springer (1988) 13–25